VESA®

Video Electronics Standards Association

920 Hillview Ct., Ste. 140
Milpitas, CA 95035

Phone: (408) 957-9270
FAX: (408) 957-9277

VESA Advanced Feature Connector (VAFC) Software Interface Standard

Version: 1.0
Revision Date: March 30, 1994

Purpose

To standardize an open software interface for the Advanced Feature Connector system for transferring pixel data between graphics and video subsystems. The interface will initially cover the Microsoft Windows Family of operating systems, but can easily later be ported to other operating environments needed. This document describes the architecture, theory, interface, and baseline operation that allows users to interchange VAFC based products from different manufacturers and have them seamlessly work together in the highest performance mode possible without having to purchase new software.

Intellectual Property

Trademarks

All trademarks used in this document are property of their respective owners.
VESA, VAFC Video Electronics Standards Association

Patents

VESA proposal and standards documents are adopted by the Video Electronics Standards Association without regard to whether their adoption may involve patents on articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the proposal or standards document.

Support for this Specification

If you have a product which incorporates VAFC, you should ask the company that manufactured your product for assistance. If you are a display or controller manufacturer, VESA can assist you with any clarification you may require. All questions must be in writing to VESA, in order of preference

Email: support@vesa.org
FAX: 408-957-9277
US Mail:
VESA
920 Hillview Ct., Ste. 140
Milpitas, CA 95035

VAFC Software Workgroup Members

Any industry standard requires input from many sources. The people listed below were members of the VAFC Software Workgroup of the VESA Advanced Video Interface Committee which was responsible for combining all the industry input into this proposal:

VAFC Software Workgroup Leader

Michael A. Yonker, Pixel Semiconductor

VAFC Software Workgroup Members

Jim Anderson, Digital Video Resources
Ed Callway, ATI Technologies
Daniel Daum, Cirrus Logic
Rod Dewell, Excalibur Solutions
Michael Eskin, Brooktree Corporation
Richard Hudson, VideoLogic
Scott Hung, Weitek
Peter Monnes, IBM Corporation
Frank Schwartz, MainStream

Table of contents

1.0 Introduction and Scope

The VAFC connector, defined in the VESA VAFC version 1.0 Hardware Specification, allows for high bandwidth point-to-point connections between the video and graphics subsystems. To take advantage of the high bandwidth that the VAFC specification outlines, a software mechanism must exist to negotiate the highest performance mode across the bus.

With the wide variety of video and graphics hardware that can be connected by this advanced connector, it is the goal of this document to propose an interface between the two subsystems that will service all types of video and graphics devices. The software driver interface that negotiates the bus must be generic so that any vendor's video subsystem will be able to communicate with any vendor's graphics subsystem. Still, the interface should have mechanisms to allow the vendors to take advantage of extra hardware capabilities without sacrificing connectability.

It is also important that the interface be seamless to existing applications and integrate well in today's operating environments. Software vendors and operating system vendors will not write new applications and operating environments for the connector, and users will not buy new software for the connector. This software must enable VAFC hardware in today's operating environments with today's software.

Today, the graphics subsystem is a well-defined area. It is hardware that is bundled with computers. The video subsystem is much harder to quantify, though. It is usually an upgrade to an existing computer system and may take various shapes: compressors/decompressors, video teleconferencing subsystems, video capture or playback boards, video overlay, etc. With these differing devices come different types of drivers. In all cases, however, the graphics subsystem remains a constant.

With this in mind, this document will attempt to propose a simple, generic mechanism to setup the VAFC bus under the Windows 3.1 and Windows NT™ 3.1 environments. This mechanism will require a VAFC compliant driver, provided with the graphics subsystem, using the Windows installable driver interface defined for multimedia devices. The video subsystem driver(s) will attempt to configure the VAFC driver through a standard interface until it finds a set of parameters that both sides can use. The video subsystem will start at the highest configuration possible (for its own subsystem) for each VAFC parameter and step-down each time the VAFC driver returns that it can not support the parameter or move on to the next parameter if it can. This will result in the highest performance possible based on the two devices.

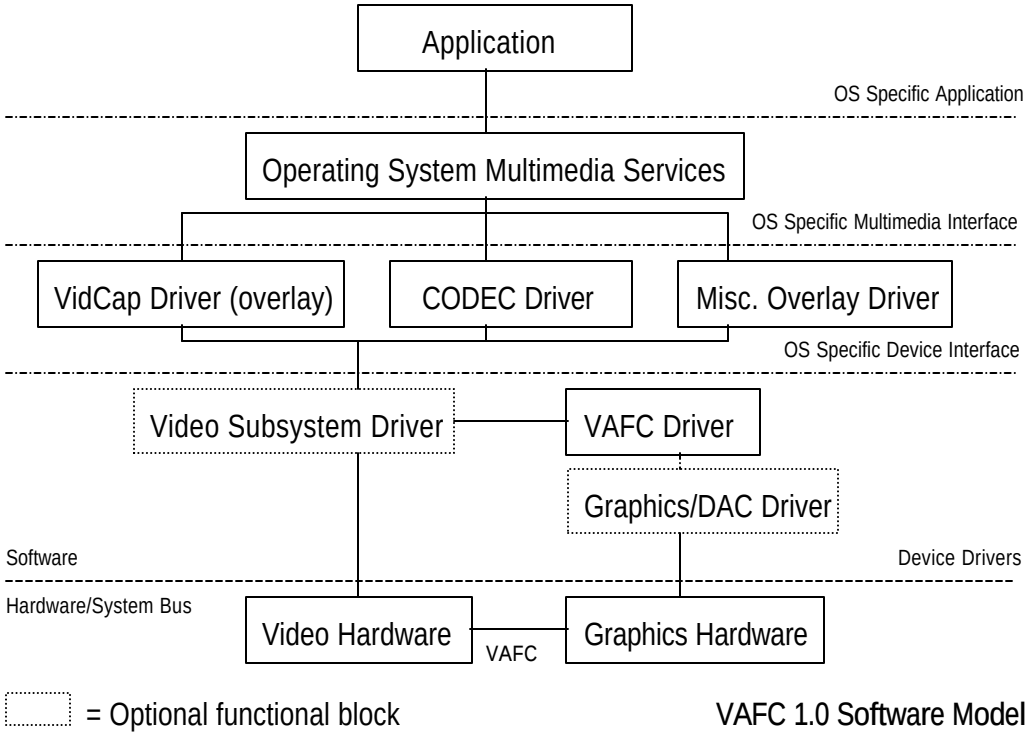
2.0 Design Features

The VAFC software interface must have the following design goals:

1. It must be open, allowing independently developed video and graphics subsystems to negotiate the configuration of the VAFC connector bus.
2. The software mechanism must not require any other hardware features that are not outlined in the VAFC specification.
3. It must be transparent to the end-user and end-user applications. End users will be able to take advantage of the new connector with current versions of their applications.
4. It must allow complete flexibility of hardware on both ends of the connectors within the limits of the VAFC specification. The software should not limit the hardware.
5. It should **not** require a ROM BIOS interface since the video subsystem has no ROM BIOS nor ROM BIOS support, and the ROM BIOS interfaces do not play well with current operating system designs.
6. The design should be simple. It should not require tremendous amounts of code or design effort to complete.
7. The design should support, initially, Microsoft Windows 3.1 and Microsoft Windows NT™ 3.1.

3.0 Design

The VAFC driver will be a Windows installable driver that is supplied and installed with the graphics subsystem. This design is compatible with other Windows multi-media drivers, and allows for a standard driver interface to be exploited, and for standard common installation/removal techniques to be used. The Windows installable driver interface is also message-based, which is preferable in today's event driven operating systems. This architecture extends to Windows NT™ 3.1. Although, the installable driver might be able to do its own I/O and IRQ handling in Windows 3.1, it will require a kernal mode driver in Windows NT™ 3.1 to perform these priveleged operations. Most likely, however, this kernal mode driver will already exist for the subsystem, and the installable driver will be architected with this in mind.



VAFC 1.0 Software Model

The video subsystem has various system drivers that can exploit the VAFC interface, and some applications which do not fit into the current architecture of MCIAVI also might want to use this resource. These drivers/custom applications (here on specified as *drivers*) are required to open the VAFC driver, communicate with the driver through the standard driver message interface to configure it for the best performance, and close the driver when the driver has finished using the VAFC bus.

The following sections explain how the video subsystem driver is expected to communicate with the VAFC driver, and correspondingly, how the VAFC driver is expected to behave.

3.1 General Operation

This section will briefly outline the programming mechanics involved in Windows installable drivers. For a more detailed explanation of the subject, refer to the online help that comes with the Microsoft Window 3.1 Software Development Kit (SDK).

The VAFC driver must be specified with the vesa.afc= key in the [drivers] section of the SYSTEM.INI file as shown below:

.

```
.
.
[drivers]
.
.
.
vesa.afc=my_vacf.drv
.
.
.
```

The video subsystem loads the VAFC driver by calling *OpenDriver* with the key name of the VAFC driver, vesa.afc:

```
.
.
.
/*
 * The VAFC hardware driver can be opened when the video subsystem driver
loads.
 * This will cause Windows to call the driver's DRV_LOAD, DRV_ENABLE, and
DRV_OPEN
 * messages.
 */
if (!hVAFCDriver)
{
  if ((hVAFCDriver = OpenDriver ((LPSTR) "vesa.afc", NULL, 0L)) == NULL)
  {
    .
    .
    .
  }
}
.
.
.
```

Once the VAFC driver is open, the video subsystem calls *SendDriverMessage* to communicate with the driver, using the handle that it returned from the *OpenDriver* call:

```
.
.
.
/*
 * If we have loaded the VAFC driver, try to query the driver for its
capabilities.
 */
if (hVAFCDriver)
{
  if (SendDriverMessage(hVAFCDriver, DRV_VAFC_GET_DEVCAPS, &DevCaps, 0) ==
FALSE)
  {
    if (SendDriverMessage(hVAFCDriver, DRV_VAFC_GET_ERROR, &dwError, 0) ==
FALSE)
    {
      /*
       * We must not be really communicating with the driver...fatal error!
       */
    }
    .
```

```
      .
      .
    }
  }
  .
  .
  .
```

It is important to point out that SendDriverMessage will only return non-zero (TRUE) and zero (FALSE). If the message returns zero, a special message exists to retrieve the appropriate error code for the failure.

The VAFC compliant video subsystem will call *CloseDriver* when it wants to unload the VAFC driver support:

```
.
.
.
/*
 * If we have loaded the driver, then close it.  This will cause Windows to
call the
 * drivers' DRV_CLOSE, DRV_DISABLE, and DRV_FREE messages.
 */
if (hVAFCDriver)
{
  CloseDriver (hVAFCDriver, 0L, 0L);
  /*
   * Initialize the handle again...
   */
  hVAFCDriver = NULL;
}
.
.
.
```

3.1 Driver interface

The installable driver interface is message based. A range of messages from DRV_RESERVED (defined in WINDOWS.H) to DRV_USER (also defined in WINDOWS.H) are available for usage by an installable driver. Each message has specific input and output parameters outlined in the standard VAFC header file, VESA_AFC.H, defined in Appendix A (pp. A-9).

This header file provides a common way for video subsystem drivers to access the VAFC functionality and a API guideline for VAFC drivers to follow. This header file should not be modified by individual companies. Mechanisms are available for vendor specific expansion to the base software specification, and these enhancements should be provided in seperate header files.

The video subsystem driver uses the *SendDriverMessage* function to communicate with the VAFC driver once the driver has been opened. The following entry points are defined for required support:

Message	Description
DRV_VAFC_GET_DEVCAPS	Gets the vendor and device information for the VAFC graphics subsystem.
DRV_VAFC_GET_COLOR_SPACE	Gets the current color space format of the connector.
DRV_VAFC_SET_COLOR_SPACE	Sets the color space format of the connector.
DRV_VAFC_GET_BUS_WIDTH	Gets the current bus transfer width of the connector.
DRV_VAFC_SET_BUS_WIDTH	Sets the bus transfer width of the connector.
DRV_VAFC_GET_DATA_MODE	Gets the current data mode of the connector.

DRV_VAFC_SET_DATA_MODE	Sets the data mode of the connector.
DRV_VAFC_GET_CLOCK_MODE	Gets the current clock mode of the connector.
DRV_VAFC_SET_CLOCK_MODE	Sets the clock mode of the connector.
DRV_VAFC_GET_OVERLAY_COLOR_INFO	Gets the overlay color information currently used by the graphics subsystem.
DRV_VAFC_SET_OVERLAY_COLOR_INFO	Sets the overlay color information used by the graphics subsystem.
DRV_VAFC_GET_GRAPHICS_INFO	Gets some miscellaneous graphics information from the graphics subsystem based on the current mode of graphics operation.
DRV_VAFC_GET_WINDOW_RECT	Gets the window coordinates for the display window.
DRV_VAFC_SET_WINDOW_RECT	Sets the window coordinates for the display window.
DRV_VAFC_QUERY_COLOR_SPACES	Gets all available video color spaces from the VAFC driver.
DRV_VAFC_GET_EXTENDED_SETUP	Gets the current extended mode parameters.
DRV_VAFC_SET_EXTENDED_SETUP	Sets the extended mode parameters.
DRV_VAFC_GET_PREFERRED_SETUP	Gets the preferred setup of the VAFC subsystem from the graphics subsystem.
DRV_VAFC_GET_ERROR	Gets the last error code and clears it.

3.1.1 Getting the vendor and device information

The DRV_VAFC_GET_DEVCAPS message is used by the video subsystem driver to determine the vendor of the graphics subsystem, the type of product, and the respective revision of that product.

The video subsystem might use this structure to exploit vendor specific capabilities of the VAFC driver. By determining who the vendor is, the video subsystem can use that vendor's specific calls to configure and use extended capabilities.

3.1.2 Configuring the color space format

The DRV_VAFC_SET_COLOR_SPACE and DRV_VAFC_GET_COLOR_SPACE messages are used by the video subsystem to manage the color space format of the data transferred across the connector.

If the graphics subsystem does not support a requested format, it should return zero and set an VAFCERR_NOSUPPORT_COLORSPACE error code.

3.1.3 Configuring the bus transfer width

The DRV_VAFC_SET_BUS_WIDTH and DRV_VAFC_GET_BUS_WIDTH messages are used by the video subsystem to manage the bus width of the data transferred across the connector. The video subsystem can set the bus width to 8-bit, 16-bit, or 32-bit mode.

If the graphics subsystem does not support the requested bus width, then it should return zero and set an VAFCERR_NOSUPPORT_BUSWIDTH error code.

3.1.4 Configuring the data mode

The DRV_VAFC_SET_DATA_MODE and DRV_VAFC_GET_DATA_MODE messages are used by the video subsystem to manage data mode (asynchronous or synchronous) of the data transferred across the connector.

If the graphics subsystem does not support the requested data mode, then it should return zero and set an VAFCERR_NOSUPPORT_DATAMODE error code.

3.1.5 Configuring the clocking mode

The DRV_VAFC_SET_CLOCK_MODE and DRV_VAFC_GET_CLOCK_MODE messages are used by the video subsystem to manage the clocking mode of the data transferred across the connector.

If the graphics subsystem does not support the requested clocking mode, then it should return zero and set an VAFCERR_NOSUPPORT_CLOCKMODE error code.

3.1.6 Configuring the overlay color key

The DRV_VAFC_SET_OVERLAY_COLOR_INFO and DRV_VAFC_GET_OVERLAY_COLOR_INFO messages are used by the video subsystem to manage the overlay color key in the graphics subsystem frame buffer.

If the graphics subsystem does not support overlay color keying, it should return zero and set the VAFCERR_NOSUPPORT_OVERLAYKEY error code.

In 8-bit palettized modes, the video driver chooses a palette index, and in true color modes, the driver chooses an RGB value compatible with the current graphics subsystem frame buffer color space format (which can be derived from the DRV_VAFC_GET_GRAPHICS_INFO message).

It is important to note that this message does not flood the overlay area with the overlay color key, but rather only sets a color or index as the overlay color key. It is the responsibility of the application or driver for the video subsystem to paint the graphics rectangle with the negotiated color or index.

3.1.7 Configuring the window coordinates

The DRV_VAFC_SET_WINDOW_RECT message is used by the video subsystem to notify the graphics subsystem driver of the position and size of the video window. The DRV_VAFC_GET_WINDOW_RECT message gets the current position and size of the video window.

If the graphics subsystem does not support window positioning or sizing, then it should ignore these messages.

3.1.8 Getting the current graphics information

The DRV_VAFC_GET_GRAPHICS_INFO message determines miscellaneous information about the current graphics subsystem mode, like border widths (in pixels), graphics color space formats, and sync polarities. This information should be re-acquired every time the video subsystem is initialized or when the graphics mode changes.

3.1.9 Getting the available video color spaces

The DRV_VAFC_QUERY_COLOR_SPACES message retrieves all of the available video data color space formats compatible with the graphics subsystem. This can be used by the video subsystem during initialization to choose the optimal video color space for the data sent across the connector. Since the video color space format is sometime a user-chosen option, this message allows the video driver to better manage the video's color space.

The video subsystem passes a pointer to an array of long integers and a value representing the size of the array, in long integers. The graphics subsystem fills the array with all the available color space format codes or until the array is full.

If all the formats can not be written in the array (not enough space), the graphics subsystem should return zero and set the VAFCERR_INCOMPLETE_LIST error code.

3.1.10 Configuring VAFC extended mode options

In VAFC extended mode, the DRV_VAFC_SET_EXTENDED_MODE and DRV_VAFC_GET_EXTENDED_MODE message configure options like the FIFO depth and full level.

If the graphics subsystem does not support VAFC extended modes or these commands, it should return zero and set the VAFCERR_NOSUPPORT_EXTENDED_MODE error code.

3.1.11 Getting the graphics subsystem's preferred setup

The DRV_VAFC_GET_PREFERRED_SETUP retrieves the preferred or optimal setup of the VAFC connector, with respect to the graphics subsystem. This is used by the video subsystem to get a base set of parameters to setup the graphics subsystem using the VAFC messages.

3.1.12 Getting the VAFC error code

The DRV_VAFC_GET_ERROR retrieves the last error code. If no error has occurred, then the graphics driver returns VAFCERR_NONE. If an error has occurred, then the VAFC driver should return that error code, and clear the error code internally.

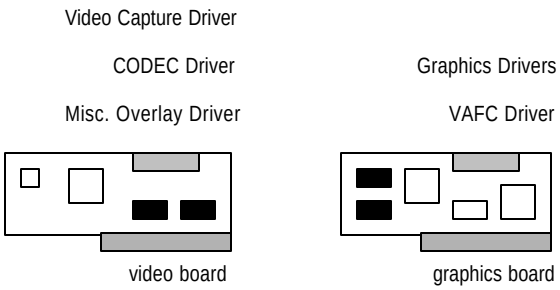
If this call returns zero (FALSE), then that must be considered a fatal error, and the video subsystem driver should stop all communication with the VAFC driver.

3.2 Deliverables

In this complex equations of boards and drivers, it is easy to confuse what driver comes with what board, and how all the pieces fit together in harmony. It is the goal of this specification, as stated earlier, to make this complexity transparent to the application and, even more so, the user.

The VAFC driver described in this specification is a **graphics subsystem deliverable**. It comes with the motherboard in systems that the VAFC graphics subsystem is "down." It also comes with VAFC graphics add-in boards.

When the user installs the graphics drivers for this board, or when the computer manufacturer pre-loads the system software, this driver is copied to the fixed disk driver (most likely in the WINDOWS\SYSTEM directory). Then the installation program updates the SYSTEM.INI file and the CONTROL.INI files.



Example Deliverables

Video subsystem deliverables depend heavily on the type of video subsystem hardware that is being provided. It could range from video capture drivers with overlay capability to CODEC drivers to custom overlay drivers for complex video teleconferencing systems. It is out of the scope of this document to discuss the myriad of video subsystem options and deliverables.

NOTE: Even in the *graphics-out* case where the graphics streams data to the video board, the VAFC driver is a graphics deliverable, and the video driver(s) contain the logic to program the appropriate VAFC hardware on their side of the connector. The picture above remains the same.

Appendix A: VESA_AFC.H Header File

```
/*****************************************************************************
 *         Copyright (c) 1993 Video Electronics Standards Association (VESA).
 *****************************************************************************
 *              VESA  Advanced Feature Connector Interface for Windows
 *****************************************************************************
 *
 * VESA_AFC.H - This header file defines the interface that a VAFC-compliant graphics
 * subsystem must supply.  It contains the VAFC standard preprocessor definitions,
 * type and structure definitions, and API messages.
 *
 *****************************************************************************/
#ifndef _VESA_AFC
#define _VESA_AFC

/* Status codes ------------------------------------------*/

#define VAFCERR_NONE                0 /* No errors have occurred */
#define VAFCERR_GENERAL_FAILURE     1 /* General failure */
#define VAFCERR_NOSUPPORT_COLORSPACE 2 /* Color space not supported */
#define VAFCERR_NOSUPPORT_BUSWIDTH  3 /* Bus width configuration not supported */
#define VAFCERR_NOSUPPORT_ESCAPECODE 4 /* Escape code not supported */
#define VAFCERR_NOSUPPORT_CLOCKING  5 /* Clocking mode not supported */
#define VAFCERR_NOSUPPORT_DATAMODE  6 /* Does not support synch/asynch data mode */
#define VAFCERR_NOSUPPORT_OVERLAYKEY 7 /* Graphics subsystem does not support overlay */
                                       /*  keying - overlay key ignored               */
#define VAFCERR_NOSUPPORT_POSITION  8 /* Graphics subsystem does not support new video */
                                      /* position */
#define VAFCERR_NOSUPPORT_EXTENDED_MODE 9 /* The graphics subsystem does not support the VAFC */
                                          /* extended mode */
#define VAFCERR_INVALID_OVERLAY_KEY  10 /* The chosen overlay color key is invalid for the */
                                        /* current graphics color space format */
#define VAFCERR_INVALID_POSITION    11 /* The chosen window position is invalid */
#define VAFCERR_INVALID_SIZE        12 /* The chosen window size is invalid */
#define VAFCERR_INCOMPLETE_LIST     13 /* The color space format list is incomplete */
#define VAFCERR_INVALID_STRUCTURE   14 /* The passed structure's size is invalid for this */
                                       /* command - too small */
#define VAFCERR_NOT_WORD_BOUNDRY    15 /* The video position/size is not on a word (16-bit) boundry */
#define VAFCERR_NOT_DWORD_BOUNDRY   16 /* The video position/size is not on a double word boundry */
#define VAFCERR_NOT_QWORD_BOUNDRY   17 /* The video position/size is not on a quad word boundry */
#define VAFCERR_NOT_DQWORD_BOUNDRY  18 /* The video position/size is not on double-quad word boundry */
#define VAFCERR_MAX_ERRORS          19 /* Number of error codes, for string table manipulation */

/* Vendor string length ----------------------------------*/

#define MAX_VENDOR_LEN              63 /* Maximum characters in the VESA vendor string */

/* VAVI Devices ------------------------------------------*/

#define VIDEO_INPUT        0x0001 /* video input device - UNUSED */
#define GRAPHIC_PORT       0x0002 /* graphics device - UNUSED */
#define VIDEO_COMPRESSOR   0x0004 /* Video compressor - UNUSED */
#define VIDEO_DECOMPRESSOR 0x0008 /* Video decompressor - UNUSED */
#define VAFC_DEVICE        0x0010 /* VAFC device - ALWAYS use this code!!! */
#define ALL_DEVICES        0xFFFF /* All VAVI devices - UNUSED */

/* Device Capabilities -----------------------------------*/

#define VAFC_SUPPORTS_OVERLAY_KEY    0x00000001 /* Device supports overlay color keying */
#define VAFC_SUPPORTS_OVERLAY_MASKS  0x00000002 /* Device supports overlay masks */
#define VAFC_SUPPORTS_EXTENDED_QUERY 0x00000004 /* Device supports extended mode GET call */
#define VAFC_SUPPORTS_EXTENDED_SETUP 0x00000008 /* Device supports extended mode SET call */
#define VAFC_SUPPORTS_POSITIONING    0x00000010 /* Device supports window positioning */
#define VAFC_SUPPORTS_OEM_API        0x00000020 /* Device has an OEM-specific API */

/*
 * Alignment flags - if none of the following is specified, it should be assumed that the VAFC
 * driver and device support byte alignment.
```

```
 */
#define VAFC_SUPPORTS_WORD_ALIGN     0x00000040 /* Device supports word (16-bit) boundry alignment */
#define VAFC_SUPPORTS_DWORD_ALIGN    0x00000080 /* Device supports double word boundry alignment */
#define VAFC_SUPPORTS_QWORD_ALIGN    0x00000100 /* Device supports quad word boundry alignment */
#define VAFC_SUPPORTS_DQWORD_ALIGN   0x00000200 /* Device supports double-quad word boundry alignment */

/*
 * The following device capabilities mask is used to determine the number of color space
 * formats the device supports.  This can be used when using the message,
 * DRV_VAFC_QUERY_COLOR_SPACES.
 */
#define VAFC_COLOR_SPACES_MASK       0xFF000000 /* Definition to mask the number of color spaces */


/* Vendor information block ------------------------------*/

typedef struct tagVESADEVICECAPS
{
  unsigned long dwSize;           /* Size of this structure */
  unsigned long dwStructRevID;    /* Structure revison ID  = 0x0100 */
  char szVendor[MAX_VENDOR_LEN+1];/* Vendor string - assigned by individual companies */
  unsigned long dwModelID;        /* Vendor specific Device ID */
  unsigned long dwRevID;          /* Vendor specific revison ID */
  unsigned short wDeviceType;     /* VESA registered device type */
  unsigned long dwDeviceCaps;     /* Device capabilities field */
  unsigned long dwReserved1;      /* Unused by VAFC, reserved for VMC usage */
  unsigned short wReserved2;      /* Unused by VAFC, reserved for VMC usage */
} VESA_DEVICECAPS;

typedef VESA_DEVICECAPS _far *LPVESA_DEVICECAPS;

/* Graphics information block ------------------------------*/

typedef struct tagGraphicsInfoBlock
{
  unsigned long dwSize;               /* Size of the structure */
  unsigned long dwColorSpace;         /* Graphics buffer color space format */
  unsigned short wActivePixelWidth;   /* Number of active pixels on the screen */
  unsigned short wActivePixelHeight;  /* Number of active lines on the screen */
  unsigned short wTopBorderHeight;    /* Number of lines in top border */
  unsigned short wBottomBorderHeight; /* Number of lines in bottom border */
  unsigned short wLeftBorderWidth;    /* Number of pixels in the left border */
  unsigned short wRightBorderWidth;   /* Number of pixels in the right border */
  unsigned short wHorzSyncPolarity;   /* Graphics horizontal sync polarity */
  unsigned short wVertSyncPolarity;   /* Graphics vertical sync polarity */
  unsigned short fInterlaced;         /* Interlaced status - TRUE/FALSE flag */
} VAFCGRAPHICS_INFO;

typedef VAFCGRAPHICS_INFO _far *LPVAFCGRAPHICS_INFO;

/* Extended mode information block --------------------------*/

typedef struct tagExtendedModeInfoBlock
{
  unsigned long dwSize;               /* Size of the structure */
  unsigned short wFIFOSize;           /* FIFO size, in bytes */
  unsigned short wFullLevel;          /* Full level of the FIFO, in bytes */
} VAFCEXTENDED_INFO;

typedef VAFCEXTENDED_INFO _far *LPVAFCEXTENDED_INFO;

/* Setup controls ---------------------------------------*/

/*
 * VAFCESCAPE allows vendors who control both sides of the interface to define
 * VENDOR SPECIFIC parameters.  If a parameter ANDed with VAFCESCAPE returns a
 * none-zero result, the option is VENDOR SPECIFIC.  If the VAFC driver
 * does not support VAFCESCAPE, it should return VAFCERR_NOSUPPORT_ESCAPECODE.
 *
 * Vendor specific parameters have to be handled on a vendor-by-vendor basis.  To
 * do this, it is assumed that the application and/or driver provider will get
 * the necessary VAFC vendor-specific interface documents and APIs from the vendor
```

```
 * itself.
 */
#define VAFCESCAPE     0x80000000 /* USER DEFINED */

/*
 * The following define the color space formats for data crossing the VAFC
 * connector.  The default format is VAFCRGB_8_INDEXED.  These same definitions
 * are used by the VAFC driver to specify the color space format of the graphics
 * buffer in the current graphics mode.
 */
#define VAFCRGB_8_INDEXED  0x00000001 /* standard 8 bit indexed color */
#define VAFCRGB_15A        0x00000002 /* 16 bit aRGB, organized as 1:5:5:5 */
#define VAFCRGB_16         0x00000004 /* 16 bit RGB, organized as 5:6:5 */
#define VAFCRGB_24         0x00000008 /* 24 bit RGB, organized as 8:8:8 */
#define VAFCRGB_32A        0x00000010 /* 32 bit aRGB, organized as 8:8:8:8 */
#define VAFCYUV_422        0x00000020 /* Packed YUV, organized as 4:2:2  */
#define VAFCYUV_444        0x00000040 /* Packed YUV, organized as 4:4:4  */

/*
 * The following extra color space definitions are provided as base escape code color
 * spaces.  Notice, as more color spaces are defined that they should be ORed with
 * the VAFCESCAPE code.
 */
#define VAFCRGB_8          VAFCESCAPE | 0x00000001 /* 8 bit RGB, organized as 3:3:2 */
#define VAFCYUV_411        VAFCESCAPE | 0x00000002 /* Packed YUV, organized as 4:1:1 */
#define VAFCRGB_15         VAFCESCAPE | 0x00000004 /* 16-bit RGB, organized as x:5:5:5 */

/*
 * The following define the available bus transfer widths available across the
 * VAFC connector.  The directions of the transfers are explicitly described in
 * the definitions.  The default or power up state of the H/W is a tri-stated
 * VAFCBUS_8_OUT.
 */
#define VAFCBUS_8_OUT  0x00000001 /* standard 8-bit data out (from DAC) */
#define VAFCBUS_8_IN   0x00000002 /* standard 8-bit data out (to DAC) */
#define VAFCBUS_16_OUT 0x00000004 /* 16-bit data bus (from DAC) */
#define VAFCBUS_16_IN  0x00000008 /* 16-bit data bus (to DAC) */
#define VAFCBUS_32_OUT 0x00000010 /* 32-bit data bus (from DAC) */
#define VAFCBUS_32_IN  0x00000020 /* 32-bit data bus (to DAC) */

/*
 * The following define the available clocking modes across the VESA VAFC bus.  The
 * default power up state of the connector is 1x clocking mode, VAFCCLOCK_1X.
 */
#define VAFCCLOCK_1X  0x00000001 /* 1x clock mode - dot clock provided */
#define VAFCCLOCK_2X  0x00000002 /* 2x clock mode - dot clock / 2 provided */
#define VAFCCLOCK_4X  0x00000004 /* 4x clock mode - dot clock / 4 provided; for 1280 mode support */

/*
 * The following define the available data modes across the VAFC connector.
 */
#define VAFCDATA_SYNCHRONOUS  0x00000001 /* Synchronous data mode */
#define VAFCDATA_ASYNCHRONOUS 0x00000002 /* Asynchronous data mode */

/* Preferred setup structure ------------------------------*/

typedef struct tagPreferredParameters
{
  unsigned long dwSize;  /* Size of this structure */
  unsigned long dwColorSpace; /* Color space format */
  unsigned long dwBusWidth; /* Bus width */
  unsigned long dwClockingMode; /* Clocking mode */
  unsigned long dwDataMode; /* Data mode */
} VAFCPREFERRED_PARAMETERS;

typedef VAFCPREFERRED_PARAMETERS far *LPPREFERRED_PARAMETERS;

/* Overlay controls -----------------------------------*/

/*
 * The values for the four bColorValueN members of the VAFCOVERLAY_CONTROLS
 * structure depends on the color space format of the graphics subsystem:
```

```
 *
 * If the graphics is using an 8-bit indexed color space, then the lower
 * eight bits of bColorValue0 is the overlay color index.  The rest of the
 * variable should be masked (value & 0xFFFFFF00).
 *
 * If the graphics is in a true-color modes, bColorValue0 is the red value,
 * bColorValue1 is the green value, and bColorValue2 is the blue value.
 *
 * If the video color space format includes alpha information, then
 * bColorValue3 is the alpha value.
 *
 * For VAFC devices that support both overlay color keying and overlay
 * masking, the bMaskValueN members can be used to mask each appropriate
 * component.
 *
 * NOTE: If the graphics subsystem does not support overlay color keying
 * (it overlays based on image size and position) then the VAFC driver
 * should return VAFCERR_NOSUPPORT_OVERLAYKEY to signify that the overlay
 * key is ignored.
 */
typedef struct tagVAFCOverlayControls
{
  unsigned long dwSize;       /* Size of the structure */
  unsigned char bColorValue0; /* Meaning depends on graphics color space format */
  unsigned char bColorValue1; /* Meaning depends on graphics color space format */
  unsigned char bColorValue2; /* Meaning depends on graphics color space format */
  unsigned char bColorValue3; /* Meaning depends on graphics color space format */
  unsigned char bMaskValue0;  /* Meaning depends on graphics color space format */
  unsigned char bMaskValue1;  /* Meaning depends on graphics color space format */
  unsigned char bMaskValue2;  /* Meaning depends on graphics color space format */
  unsigned char bMaskValue3;  /* Meaning depends on graphics color space format */
} VAFCOVERLAY_CONTROLS;

typedef VAFCOVERLAY_CONTROLS far *LPVAFCOVERLAY_CONTROLS;

/* Interface messages -----------------------------------*/

/*
 * Message : DRV_VAFC_GET_DEVCAPS
 * LParam1 : (LPARAM)(LPVESA_DEVICECAPS) &lpDevCaps;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver passes a far pointer to its VESA_DEVICECAPS
 * structure. DRV_VAFC_GET_DEVCAPS will load the structure with valid data
 * and return non-zero, or else it will return zero and set the appropriate
 * VAFC error code.
 */
#define DRV_VAFC_GET_DEVCAPS       DRV_RESERVED

/*
 * Message : DRV_VAFC_GET_COLOR_SPACE
 * LParam1 : (LPARAM) (unsigned long far *) &dwColorSpace;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver sends the DRV_VAFC_GET_COLOR_SPACE message
 * to the VAFC driver to get the current color space format across the connector.
 *
 * It should be noted that this does not define the graphics buffer mode, but
 * rather the color space mode of the data across the connector, which can be
 * independent of the graphics color space.
 */
#define DRV_VAFC_GET_COLOR_SPACE    DRV_RESERVED + 1

/*
 * Message : DRV_VAFC_SET_COLOR_SPACE
 * LParam1 : (LPARAM) dwColorSpace;
```

```
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver sends the DRV_VAFC_SET_COLOR_SPACE message
 * to the VAFC driver to set the current color space format across the connector.
 *
 * If the VAFC driver can not support the requested color space it should return
 * zero and set the VAFC error code to VAFCERR_NOSUPPORT_COLORSPACE, indicating
 * that the video subsystem should request a different color space.
 *
 * It should be noted that this does not define the graphics buffer mode, but
 * rather the color space mode of the data across the connector, which can be
 * independent of the graphics color space.
 */
#define DRV_VAFC_SET_COLOR_SPACE    DRV_RESERVED + 2

/*
 * Message : DRV_VAFC_GET_BUS_WIDTH
 * LParam1 : (LPARAM) (unsigned long far *) &dwBusWidth;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver sends the DRV_VAFC_GET_BUS_WIDTH message to determine
 * the current bus transfer width across the VAFC connector.
 */
#define DRV_VAFC_GET_BUS_WIDTH    DRV_RESERVED + 3

/*
 * Message : DRV_VAFC_SET_BUS_WIDTH
 * LParam1 : (LPARAM) dwBusWidth;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver passes the desired bus width code, and if the VAFC
 * driver can support the mode, it returns non-zero, otherwise, it returns zero and
 * sets the appropriate error code.
 *
 * If the VAFC driver can not support the requested bus width it should return zero
 * and set VAFCERR_NOSUPPORT_BUSWIDTH, indicating that the video subsystem should request
 * a different bus width.
 */
#define DRV_VAFC_SET_BUS_WIDTH    DRV_RESERVED + 4

/*
 * Message : DRV_VAFC_GET_DATA_MODE
 * LParam1 : (LPARAM) (unsigned long far *) &dwDataMode;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver uses the DRV_VAFC_GET_DATA_MODE message to determine
 * the current data mode (sync/async) across the VAFC connector.
 */
#define DRV_VAFC_GET_DATA_MODE    DRV_RESERVED + 5

/*
 * Message : DRV_VAFC_SET_DATA_MODE
 * LParam1 : (LPARAM) dwDataMode;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver passes a request for a synchronous or asynchronous
 * data mode, and the VAFC driver returns VAFCERR_NONE is it is successful or
```

```
 * an appropriate error code if it fails.
 *
 * If the VAFC driver can not support the requested data mode it should return zero
 * and set VAFCERR_NOSUPPORT_DATAMODE, indicating that the video subsystem should request
 * a different data mode.
 */
#define DRV_VAFC_SET_DATA_MODE    DRV_RESERVED + 6

/*
 * Message : DRV_VAFC_GET_CLOCK_MODE
 * LParam1 : (LPARAM) (unsigned long far *) &dwClockMode;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver uses the DRV_VAFC_GET_CLOCK_MODE message to determine
 * the current clocking mode across the VAFC connector.
 *
 * NOTE: The clocking defines an implied X zoom (or replicate) by the graphics subsystem.
 * In clock divide by 2 mode (VAFCCLOCK_2X), the graphics subsystem is providing a divide
 * by two clock, which means that it will zoom the data 2X.  The video subsystem should provide
 * half the resolution that it expects to be displayed in the X direction, and it should
 * probably provide 2Y the resolution to maintain the window apect ratio.
 */
#define DRV_VAFC_GET_CLOCK_MODE    DRV_RESERVED + 7

/*
 * Message : DRV_VAFC_SET_CLOCK_MODE
 * LParam1 : (LPARAM) dwClockMode;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver uses the DRV_VAFC_SET_CLOCK_MODE message to set
 * a new clock mode to be used across the VAFC bus.
 *
 * If the VAFC driver can not support the requested clocking mode it should return zero
 * and set VAFCERR_NOSUPPORT_CLOCKMODE, indicating that the video subsystem should request
 * a different clocking mode.
 *
 * NOTE: The clocking defines an implied X zoom (or replicate) by the graphics subsystem.
 * In clock divide by 2 mode (VAFCCLOCK_2X), the graphics subsystem is providing a divide
 * by two clock, which means that it will zoom the data 2X.  The video subsystem should provide
 * half the resolution that it expects to be displayed in the X direction, and it should
 * probably provide 2Y the resolution to maintain the window apect ratio.
 */
#define DRV_VAFC_SET_CLOCK_MODE    DRV_RESERVED + 8

/*
 * Message : DRV_VAFC_GET_OVERLAY_COLOR_INFO
 * LParam1 : (LPARAM) (LPVAFCOVERLAY_CONTROLS) &lpOverlay;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver passes a far pointer to the VAFCOVERLAY_CONTROLS
 * structure.  This values returned in this structure vary depending on the graphics
 * color space format.  The VAFC driver returns non-zero, it is successful or zero if it
 * fails.
 */
#define DRV_VAFC_GET_OVERLAY_COLOR_INFO    DRV_RESERVED + 9

/*
 * Message : DRV_VAFC_SET_OVERLAY_COLOR_INFO
 * LParam1 : (LPARAM) (LPVAFCOVERLAY_CONTROLS) &lpOverlay;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
```

```
 *
 * The video application or driver passes a far pointer to the VAFCOVERLAY_CONTROLS
 * structure.  The values passed in this structure vary depending on the graphics
 * color space format.  The VAFC driver should return non-zero if it is successful
 * or zero and set an appropriate error code if it fails.
 *
 * NOTE: If the graphics subsystem does not support overlay color keying (it overlays
 * based on image size and position) then the VAFC driver should return zero and set
 * VAFCERR_NOSUPPORT_OVERLAYKEY to signify that the overlay key is ignored.
 */
#define DRV_VAFC_SET_OVERLAY_COLOR_INFO    DRV_RESERVED + 10

/*
 * Message : DRV_VAFC_GET_GRAPHICS_INFO
 * LParam1 : (LPARAM) (LPVAFCGRAPHICS_INFO) &lpGraphicsInfo;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver passes a far pointer to a VAFCGRAPHICS_INFO structure,
 * and the VAFC driver sets the structure members to the appropriate values for the
 * current video mode.
 */
#define DRV_VAFC_GET_GRAPHICS_INFO    DRV_RESERVED + 11

/*
 * Message : DRV_VAFC_GET_WINDOW_RECT
 * LParam1 : (LPARAM) (RECT FAR *) lpWindowRect;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver passes a far pointer to a RECT structure (defined in
 * WINDOWS.H).  The VAFC driver returns the coordinates of the display window.  The VAFC
 * driver returns non-zero is it is successful or zero and sets an appropriate error code
 * if it fails.
 */
#define DRV_VAFC_GET_WINDOW_RECT    DRV_RESERVED + 12

/*
 * Message : DRV_VAFC_SET_WINDOW_RECT
 * LParam1 : (LPARAM) (RECT FAR *) lpWindowRect;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver passes a far pointer to a RECT structure (defined in
 * WINDOWS.H).  The VAFC driver sets its internal structures and registers to reflect
 * the new positioning/sizing.  The VAFC driver returns non-zero is it is successful
 * or zero and sets an appropriate error code if it fails.
 *
 * NOTE: In systems where the position/sizing are controlled in the video subsystem, this
 * can act as a notification.  In systems where the positioning/sizing are controlled in
 * the graphics subsystem, this can act a command.
 */
#define DRV_VAFC_SET_WINDOW_RECT    DRV_RESERVED + 13


/*
 * Message : DRV_VAFC_QUERY_COLOR_SPACES
 * LParam1 : (LPARAM) (unsigned long far *) &ColorSpaceArray;
 * LParam2 : (LPARAM) dwSizeOfArray;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver passes a far pointer to an array of unsigned longs and
 * size.  The VAFC driver fills in the array with the available video color spaces
 * supported by the graphics subsystem.  The size parameter is used to define the maximum
```

```
 * size of the array.
 */
#define DRV_VAFC_QUERY_COLOR_SPACES    DRV_RESERVED + 14


/*
 * Message : DRV_VAFC_GET_EXTENDED_SETUP
 * LParam1 : (LPARAM) (LPVAFCEXTENDED_INFO) &lpExtendedInfo;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver uses this message to get the current extended
 * mode parameters.  If the graphics subsystem does not support extended mode,
 * then it should return zero and set VAFCERR_NOSUPPORT_EXTENDED_MODE.
 */
#define DRV_VAFC_GET_EXTENDED_SETUP    DRV_RESERVED + 15


/*
 * Message : DRV_VAFC_SET_EXTENDED_SETUP
 * LParam1 : (LPARAM) (LPVAFCEXTENDED_INFO) &lpExtendedInfo;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver uses this message to set the VAFC extended
 * mode parameters.  If the graphics subsystem does not support extended mode,
 * then it should return zero and set VAFCERR_NOSUPPORT_EXTENDED_MODE.
 */
#define DRV_VAFC_SET_EXTENDED_SETUP    DRV_RESERVED + 16

/*
 * Message : DRV_VAFC_GET_PREFERRED_SETUP
 * LParam1 : (LPARAM) (LPPREFERRED_PARAMETERS) &lpParams;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.  Use the DRV_VAFC_GET_ERROR
 *           message to get the error code for the failure.
 *
 * The video application or driver can use this message when initializing to speed
 * up the optimal selection of VAFC parameters.  The graphics subsystem
 * initializes this array with its preferred setup parameters.
 */
#define DRV_VAFC_GET_PREFERRED_SETUP    DRV_RESERVED + 17

/*
 * Message : DRV_VAFC_GET_ERROR
 * LParam1 : (LPARAM) (unsigned long far *) &dwError;
 * LParam2 : (LPARAM) 0L;
 *
 * LRESULT : Non-zero if successful, or zero if the call failed.
 *
 * The video application or driver uses this message to retrieve the last error code.  The
 * VAFC driver should clear the error code when this message is processed.
 */
#define DRV_VAFC_GET_ERROR    DRV_RESERVED + 18


/*
 * VENDOR-SPECIFIC MESSAGE BLOCK
 * -----------------------------
 * The following define the block of available vendor-specific messages available for
 * hardware vendors to add messages to support extra, non-VAFC-standard features in
 * the same VAFC driver.  A video subsystem driver can determine what vendor and device
 * using the DRV_VAFC_GET_DEVCAPS message to take advantage of extra features.
 *
 * NOTE: The video subsystem driver would have to know the messages available from the
 * specific VAFC driver.  This information would be provided by the VAFC driver supplier,
 * not VESA.
```

```
 */
#define DRV_VAFC_VENDOR_START    DRV_RESERVED + 0x1000
#define DRV_VAFC_VENDOR_END      DRV_RESERVED + 0x1300

#endif /* _VESA_AFC_H */
```

Appendix B: VAFC Messages

DRV_VAFC_GET_BUS_WIDTH

```
Message : DRV_VAFC_GET_BUS_WIDTH
LParam1 : (LPARAM) (unsigned long far *) &dwBusWidth;
```

The video application or driver sends the DRV_VAFC_GET_BUS_WIDTH message to determine the current bus transfer width across the VAFC connector. It should initialize the variable with the current bus width code and return TRUE, if successful. Otherwise it should return FALSE and set the appropriate VAFC error code.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_GET_CLOCK_MODE

```
Message : DRV_VAFC_GET_CLOCK_MODE
LParam1 : (LPARAM) (unsigned long far *) &dwClockMode;
```

The video application or driver uses the DRV_VAFC_GET_CLOCK_MODE message to determine the current clocking mode across the VAFC connector. It should initialize the variable with the current clocking mode code and return TRUE, if successful. Otherwise it should return FALSE and set the appropriate VAFC error code.

NOTE: The clocking defines an implied X zoom (or replicate) by the graphics subsystem. In clock divide by 2 mode (VAFCCLOCK_2X), the graphics subsystem is providing a divide by two clock, which means that it will zoom the data 2X. The video subsystem should provide half the resolution that it expects to be displayed in the X direction, and it should probably provide 2Y the resolution to maintain the window apect ratio.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_GET_COLOR_SPACE

```
Message : DRV_VAFC_GET_COLOR_SPACE
LParam1 : (LPARAM) (unsigned long far *) &dwColorSpace;
```

The video application or driver sends the DRV_VAFC_GET_COLOR_SPACE message to the VAFC driver to get the current color space format across the connector. It should initialize the variable with the current color space format code and return TRUE, if successful. Otherwise it should return FALSE and set the appropriate VAFC error code.

It should be noted that this does not define the graphics frame buffer color space format, but rather the color space format of the data across the connector, which can be independent of the graphics color space.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_GET_DATA_MODE

```
Message : DRV_VAFC_GET_DATA_MODE
LParam1 : (LPARAM) (unsigned long far *) &dwDataMode;
```

The video application or driver uses the DRV_VAFC_GET_DATA_MODE message to determine the current data mode (sync/async) across the VAFC connector. It should initialize the variable with the current data mode code and return TRUE, if successful. Otherwise it should return FALSE and set the appropriate VAFC error code.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_GET_DEVCAPS

```
Message : DRV_VAFC_GET_DEVCAPS
LParam1 : (LPARAM)(LPVESA_DEVICECAPS) &lpVendor;
```

The video application or driver passes a far pointer to its VESA_DEVICECAPS structure. DRV_VAFC_GET_DEVCAPS will initialize the structure with valid data and return TRUE if successful. Otherwise it should return FALSE and set the appropriate VAFC error code.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_GET_ERROR

```
Message : DRV_VAFC_GET_ERROR
LParam1 : LPARAM) (unsigned long far *) &dwError;
```

The video application or driver uses this message to retrieve the last error code. The VAFC driver should clear the error code when this message is processed.

Return Value Non-zero if successful, or zero if the call failed.

DRV_VAFC_GET_EXTENDED_SETUP

```
Message : DRV_VAFC_GET_EXTENDED_SETUP
LParam1 : (LPARAM) (LPVAFCEXTENDED_INFO) &lpExtendedInfo;
```

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_GET_GRAPHICS_INFO

```
Message : DRV_VAFC_GET_GRAPHICS_INFO
LParam1 : (LPARAM) (LPVAFCGRAPHICS_INFO) &lpGraphicsInfo;
```

The video application or driver passes a far pointer to a VAFCGRAPHICS_INFO structure, and the VAFC driver sets the structure members to the appropriate values for the current video mode.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_GET_OVERLAY_COLOR_INFO

```
Message : DRV_VAFC_GET_OVERLAY_COLOR_INFO
```

```
LParam1 : (LPARAM) (LPVAFCOVERLAY_CONTROLS) &lpOverlay;
```

The video application or driver passes a far pointer to the VAFCOVERLAY_CONTROLS structure. This values returned in this structure vary depending on the graphics color space format. It should initialize the structure with the current overlay color info and return TRUE, if successful. Otherwise it should return FALSE and set the appropriate VAFC error code.

The VAFC driver returns VAFCERR_NONE, it is successful or an appropriate error code if it fails.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_GET_PREFERRED_SETUP

```
Message : DRV_VAFC_GET_PREFERRED_SETUP
LParam1 : (LPARAM) (LPPREFERRED_PARAMETERS) &lpParams;
```

The video application or driver can use this message when initializing to speed up the optimal selection of VAFC parameters. The graphics subsystem initializes this array with its preferred setup parameters.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_GET_WINDOW_RECT

```
Message : DRV_VAFC_GET_WINDOW_RECT
LParam1 : (LPARAM) (RECT FAR *) lpWindowRect;
```

The video application or driver passes a far pointer to a RECT structure (defined in WINDOWS.H). The VAFC driver returns the coordinates of the display window. It should initialize the structure with the current window coordinates and size and return TRUE, if successful. Otherwise it should return FALSE and set the appropriate VAFC error code.

The VAFC driver returns VAFCERR_NONE is it is successful or an appropriate error code if it fails.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_QUERY_COLOR_SPACES

```
Message : DRV_VAFC_QUERY_COLOR_SPACES
LParam1 : (LPARAM) (unsigned long far *) &ColorSpaceArray;
LParam2 : (LPARAM) dwSizeOfArray;
```

The video application or driver passes a far pointer to an array of unsigned longs and size. The VAFC driver fills in the array with the available video color spaces supported by the graphics subsystem. The size parameter is used to define the maximum size of the array.

If the graphics driver does not have enough space to complete its list of available color space formats, it should return zero and set the error code, VAFCERR_INCOMPLETE_LIST.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_SET_BUS_WIDTH

```
Message : DRV_VAFC_SET_BUS_WIDTH
LParam1 : (LPARAM) dwBusWidth;
```

The video application or driver passes the desired bud width code, and if the VAFC driver can support the mode, it returns VAFCERR_NONE, otherwise, it returns an appropriate error code.

If the VAFC driver can not support the requested bus width it should return VAFCERR_NOSUPPORT_BUSWIDTH, indicating that the video subsystem should request a different bus width.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_SET_CLOCK_MODE

```
Message : DRV_VAFC_SET_CLOCK_MODE
LParam1 : (LPARAM) wClockMode;
```

The video application or driver uses the DRV_VAFC_SET_CLOCK_MODE message to set a new clock mode to be used across the VAFC bus. The VAFC driver should return VAFCERR_NONE if it is successful or an appropriate error code if it fails.

If the VAFC driver can not support the requested clocking mode it should return VAFCERR_NOSUPPORT_DATAMODE, indicating that the video subsystem should request a different clocking mode.

NOTE: The clocking defines an implied X zoom (or replicate) by the graphics subsystem. In clock divide by 2 mode (VAFCCLOCK_2X), the graphics subsystem is providing a divide by two clock, which means that it will zoom the data 2X. The video subsystem should provide half the resolution that it expects to be displayed in the X direction, and it should probably provide 2Y the resolution to maintain the window apect ratio.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_SET_COLOR_SPACE

```
Message : DRV_VAFC_SET_COLOR_SPACE
LParam1 : (LPARAM) dwColorSpace;
```

The video application or driver sends the DRV_VAFC_GET_COLOR_SPACE message to the VAFC driver to get the current color space format across the connector. It should set the requested color space format for the data being transferred across the connector and return TRUE, if successful. Otherwise it should return FALSE and set the appropriate VAFC error code.

If the VAFC driver can not support the requested color space it should return VAFCERR_NOSUPPORT_COLOR_SPACE, indicating that the video subsystem should request a different color space.

It should be noted that this does not define the graphics buffer mode, but rather the color space mode of the data across the connector, which can be independent of the graphics color space.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_SET_DATA_MODE

```
Message : DRV_VAFC_SET_DATA_MODE
LParam1 : (LPARAM) wDataMode;
```

The video application or driver passes a request for a synchronous or asynchronous data mode, and the VAFC driver returns VAFCERR_NONE is it is successful or an appropriate error code if it fails.

If the VAFC driver can not support the requested data mode it should return VAFCERR_NOSUPPORT_DATAMODE, indicating that the video subsystem should request a different data mode.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_SET_EXTENDED_SETUP

```
Message : DRV_VAFC_SET_EXTENDED_SETUP
LParam1 : (LPARAM) (LPVAFCEXTENDED_INFO) &lpExtendedInfo;
```

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_SET_OVERLAY_COLOR_INFO

```
Message : DRV_VAFC_SET_OVERLAY_COLOR_INFO
LParam1 : (LPARAM) (LPVAFCOVERLAY_CONTROLS) &lpOverlay;
```

The video application or driver passes a far pointer to the VAFCOVERLAY_CONTROLS structure. The values passed in this structure vary depending on the graphics color space format. The VAFC driver should return VAFCERR_NONE if it is successful or an appropriate error code if it fails.

NOTE: If the graphics subsystem does not support overlay color keying (it overlays based on image size and position) then the VAFC driver should return VAFCERR_NOSUPPORT_OVERLAYKEY to signify that the overlay key is ignored.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

DRV_VAFC_SET_WINDOW_RECT

```
Message : DRV_VAFC_SET_WINDOW_RECT
LParam1 : (LPARAM) (RECT FAR *) lpWindowRect;
```

The video application or driver passes a far pointer to a RECT structure (defined in WINDOWS.H). The VAFC driver sets its internal structures and registers to reflect the new positioning/sizing. The VAFC driver returns VAFCERR_NONE is it is successful or an appropriate error code if it fails.

NOTE: In systems where the position/sizing are controlled in the video subsystem, this can act as a notification. In systems where the positioning/sizing are controlled in the graphics subsystem, this can act a command.

Return Value Non-zero if successful, or zero if the call failed. Use the DRV_VAFC_GET_ERROR message to get the error code for the failure.

Appendix C: VAFC Structures

VAFCEXTENDED_INFO

```
typedef struct tagExtendedModeInfoBlock
{
  unsigned long dwSize;
  unsigned short wFIFOSize;
  unsigned short wFullLevel;
} VAFCEXTENDED_INFO;
```

The VAFCEXTENDED_INFO structure contains the VAFC extended mode information. This includes the FIFO depth and full level.

Member	Description
dwSize	Size of the structure
wFIFOSize	FIFO size, in bytes
wFullLevel	Full level of the FIFO, in bytes

VAFCGRAPHICS_INFO

```
typedef struct tagGraphicsInfoBlock
{
  unsigned long dwSize;
  unsigned long dwColorSpace;
  unsigned short wActivePixelWidth;
  unsigned short wActivePixelHeight;
  unsigned short wTopBorderHeight;
  unsigned short wBottomBorderHeight;
  unsigned short wLeftBorderWidth;
  unsigned short wRightBorderWidth;
  unsigned short wHorzSyncPolarity;
  unsigned short wVertSyncPolarity;
  unsigned short fInterlaced;
} VAFCGRAPHICS_INFO;
```

The VAFCGRAPHICS_INFO structure contains miscellaneous graphics subsystem information that the various video subsystems might require, such as border widths and sync polarities.

Member	Description
dwSize	Size of the structure
dwColorSpace	Graphics buffer color space format
wActivePixelWidth	Number of active pixels on the screen
wActivePixelHeight	Number of active lines on the screen
wTopBorderHeight	Number of lines in top border
wBottomBorderHeight	Number of lines in bottom border
wLeftBorderWidth	Number of pixels in the left border
wRightBorderWidth	Number of pixels in the right border
wHorzSyncPolarity	Graphics horizontal sync polarity
wVertSyncPolarity	Graphics vertical sync polarity
fInterlaced	Interlaced status - TRUE/FALSE flag

VAFCOVERLAY_CONTROLS

```
typedef struct tagVAFCOverlayControls
{
  unsigned long dwSize;
  unsigned char bColorValue0;
  unsigned char bColorValue1;
  unsigned char bColorValue2;
  unsigned char bColorValue3;
  unsigned char bMaskValue0;
  unsigned char bMaskValue1;
  unsigned char bMaskValue2;
  unsigned char bMaskValue3;
} VAFCOVERLAY_CONTROLS;
```

The values for the four bColorValueN members of the VAFCOVERLAY_CONTROLS structure depends on the color space format of the graphics subsystem.

If the graphics is using an 8-bit indexed color space, then the lower eight bits of bColorValue0 is the overlay color index. The rest of the variable should be masked (value & 0xFFFFFF00).

If the graphics is in a true-color modes, bColorValue0 is the red value, bColorValue1 is the green value, and bColorValue2 is the blue value.

If the video color space format includes alpha information, then bColorValue3 is the alpha value.

For VAFC devices that support both overlay color keying and overlay masking, the bMaskValueN members can be used to mask each appropriate component.

Member	Description
dwSize	Size of the structure
bColorValue0	Meaning depends on graphics color space format
bColorValue1	Meaning depends on graphics color space format
bColorValue2	Meaning depends on graphics color space format
bColorValue3	Meaning depends on graphics color space format
bMaskValue0	Meaning depends on graphics color space format
bMaskValue1	Meaning depends on graphics color space format
bMaskValue2	Meaning depends on graphics color space format
bMaskValue3	Meaning depends on graphics color space format

VAFCPREFERRED_PARAMETERS

```
typedef struct tagPreferredParameters
{
  unsigned long dwSize;
  unsigned long dwColorSpace;
  unsigned long dwBusWidth;
  unsigned long dwClockingMode;
  unsigned long dwDataMode;
} VAFCPREFERRED_PARAMETERS;
```

The VAFCPREFERRED_PARAMETERS structure contains the preferred or suggested VAFC setup parameters for the graphics subsystem. This includes the base parameters: color space format, bus transfer width, clocking mode, and data mode.

Member	Description
dwSize	Size of this structure
dwColorSpace	The color space format of the data transferred across the connector.
dwBusWidth	The bus transfer width of the data transferred across the connector.
dwClockingMode	The clocking mode of the data transferred across the connector.
dwDataMode	The data mode of the data transferred across the connector.

VESA_DEVICECAPS

```
typedef struct tagVESADEVICECAPS
{
  unsigned long dwSize;
  unsigned long dwStructRevID;
  char szVendor[MAX_VENDOR_LEN+1];
  unsigned long dwModelID;
  unsigned long dwRevID;
  unsigned short wDeviceType;
  unsigned long dwReserved0;
  unsigned long dwReserved1;
  unsigned short wReserved2;
} VESA_DEVICECAPS;
```

The VESA_DEVICECAPS structure is a common structure between the VESA VAFC and VESA VMC specifications used to give basic information about the particular driver: who made it, what's it's revision and model, etc.

Member	Description
dwSize	Size of this structure
dwStructRevID	Structure revison ID
szVendor[]	Vendor string - assigned by individual companies
dwModelID	Vendor specific Device ID
dwRevID	Vendor specific revison ID
wDeviceType	VESA registered device type
dwReserved0	Unused by VAFC, reserved for VMC usage
dwReserved1	Unused by VAFC, reserved for VMC usage
wReserved2	Unused by VAFC, reserved for VMC usage

Appendix D: VAFC Errors

Error code	Description
VAFCERR_NONE	No errors have occurred
VAFCERR_GENERAL_FAILURE	General failure
VAFCERR_NOSUPPORT_COLORSPACE	The graphics subsystem does not support the requested color space format.
VAFCERR_NOSUPPORT_BUSWIDTH	The graphics subsystem does not support the requested bus width configuration.
VAFCERR_NOSUPPORT_ESCAPECODE	The graphics subsystem does not support the requested escape code.
VAFCERR_NOSUPPORT_CLOCKING	The graphics subsystem does not support the requested clocking mode.
VAFCERR_NOSUPPORT_DATAMODE	The graphics subsystem does not support the requested synch/asynch data mode
VAFCERR_NOSUPPORT_OVERLAYKEY	The graphics subsystem does not support overlay color keying - overlay key ignored
VAFCERR_NOSUPPORT_POSITION	The graphics subsystem does not support new video position
VAFCERR_NOSUPPORT_EXTENDED_MODE	The graphics subsystem does not support the VAFC extended mode.
VAFCERR_INVALID_OVERLAY_KEY	The requested overlay color key is invalid for the current graphics color space format.
VAFCERR_INVALID_POSITION	The requested window position is invalid.
VAFCERR_INVALID_SIZE	The requested window size is invalid.
VAFCERR_INCOMPLETE_LIST	The color space format list is incomplete.
VAFCERR_INVALID_STRUCTURE	The passed structure's size is invalid for this command - too small.

Appendix E: VAFC Software Installation

The following section briefly explains the different installation options for Windows installable drivers. For a more complete reference on the subject, refer to the online documentation that comes with the Microsoft Windows 3.1 Software Development Kit (SDK).

E.1 Installation Instructions

The VAFC driver described in this specification is a graphics subsystem deliverable. It comes with the motherboard in systems that the graphics subsystem is "down." It also comes with graphics add-in boards. When the user installs the graphics drivers for this board, or when the computer manufacturer pre-loads the system software, this driver is copied to the fixed disk driver (most likely in the WINDOWS\SYSTEM directory or the WINDOW directory). Then the installation program updates the SYSTEM.INI file and the CONTROL.INI files.

E.1.1 VAFC Driver Installation

The VAFC driver should be installed into the \WINDOWS\SYSTEM directory. If it is installed into another directory, its full path name can be used in the INI files to specify its location. This should be copied to the fixed disk drive when the graphics subsystem drivers are installed.

SYSTEM.INI:

The *vafc.afc=* line is added to the *[drivers]* section, along with the name (or full path) of the VAFC driver for the graphics board:

```
.
.
.
[drivers]
msvideo=my_vcap.drv
vesa.afc=my_vafc.drv    ; OR "=drive:\fullpath\my_vafc.drv" if the driver
is not
                        ;  in the SYSTEM directory or the WINDOWS directory
.
.
.
```

CONTROL.INI:

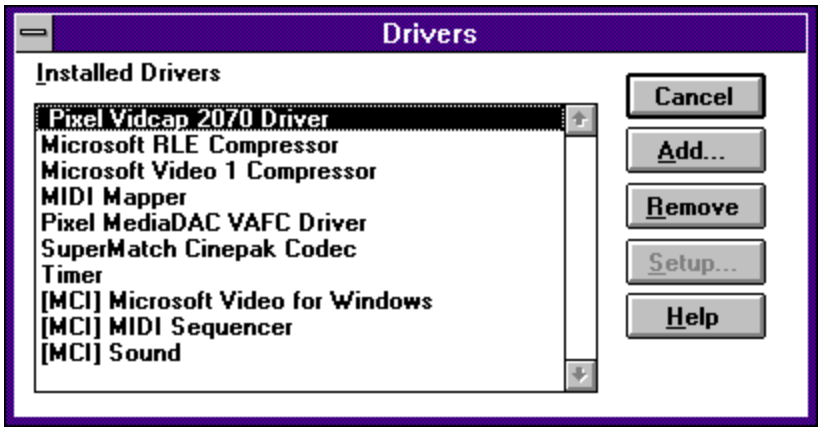
The *my_vafc.drv=* line is added to the *[drivers.desc]* section, along with an appropriate description line, which appears in the list box in the Drivers' Control Panel applet:

```
.
.
.
[drivers.desc]
my_vcap.drv=My VidCap Driver
my_vafc.drv=My VESA VAFC Driver  ; OR "drive:\fullpath\my_vafc.drv=" if the
driver
                                 ;  is not in the SYSTEM directory or the
WINDOWS
                                 ;  directory
.
.
```

.

E.1.2 User-prompted VAFC Driver Installation/Removal

Since the VAFC driver is a Window installable driver, it's installation and de-installation can be controlled by the user from the Drivers applet in the Control Panel. The user double-clicks on the Control Panel icon (to run it) and double-clicks again on the Driver icon. This will bring up the following dialog box:



The following extract from the Microsoft Corporation Windows 3.1 SDK online document explains the OEMSETUP.INF file that ships with the installable driver to tell Windows how it should install and remove the driver and its corresponding support pieces:

Contents of the OEMSETUP.INF Files

The OEMSETUP.INF file uses the same format as the Windows 3.0 SETUP.INF file with the exception of a new [Installable.Drivers] section. This section identifies the names and characteristics of each driver on the disk. Each driver entry has the following form:

```
entry=disk:filename, type(s), description, VxD(s), default_params
```

Note that the elements that compose a driver entry are separated by commas. Comments are delimited by semicolons; all characters following a semicolon are considered part of the comment string. Following are the elements that compose a driver entry:

Element	Description
entry	Identifies the driver. This string must be unique.
disk	Specifies the disk number for the disk that contains the driver. This entry corresponds to an entry in the [disks] section of SETUP.INF.
filename	Specifies the name of the file that contains the driver.
type(s)	Specifies the driver type.
description	Describes the driver. This string appears in the dialog box displayed by the Drivers Control Panel application.
VxDs	Identifies any VxDs required by the driver. (For a description of the manner in which multiple VxD names are parsed, see the Microsoft Windows Virtual Device Adaptation Guide.)
default_params	Specifies default parameters for the driver. Additional options are appended to the driver entry in the [drivers] section of SYSTEM.INI.

If you create an OEMSETUP.INF file to distribute with your driver, it must include the [disks] and [Installable.Drivers] sections. For example, the following entries could be used in an OEMSETUP.INF file for a prototype installable driver:

```
.
.
.
[disks]
1; Numeric mappings for disk titles

1 = ., "Sample Distribution Disk 1"

[Installable.Drivers]
; The installable drivers section is unique to the drivers application.
; It is parsed with comma-separated fields.

prototype=1:proto.drv,"ampl,freq","Sample scope driver","1:VXDA.386"
.
.
.
```

The Drivers Control Panel application may need to copy files that support your driver. If any of these files are not VxDs, include a section in the SYSTEM.INI file listing them. Use the entry (that is, prototype) as the name of this new section. For example, if the prototype driver has an additional file called POWERSRC.DLL, include the following section:

```
.
.
.
[prototype]
; Keyname sections can be created for dependent files.  All
; dependent files will be copied directly to the system directory.

1:POWERSRC.DLL
.
.
.
```

Drivers Control Panel Application

The Drivers Control Panel application installs, configures, and removes drivers. When started, the Drivers Control Panel application displays the following dialog box.

The Installed Drivers list box displays the description strings of the installed drivers. The installed drivers are determined by examining the [drivers] and [mci] sections of the SYSTEM.INI file. The description strings are cached in the [drivers.description] section of the CONTROL.INI file to reduce delays in finding and loading them. If a description string does not match an installed driver, the application searches the MMSETUP.INF file and then the header of the driver file to obtain the description string. A scroll bar appears in the list box if there are more drivers than can be displayed. The following buttons are found in the Control Panel dialog box:

Button	Result when chosen
OK	Exits the dialog box and makes any changes permanent.
Cancel	Exits the dialog box. The application ignores any requests to install or remove drivers made during the session. Any configuration changes made during the session are retained because they are done by the driver.

Remove	Removes the information about the selected driver from the SYSTEM.INI file. When removing drivers, the Control Panel application sends the DRV_REMOVE message to the driver if there is only one entry in the SYSTEM.INI file for it.
Setup	Applies only to configurable drivers. When the user selects a driver in the list box, the application opens the driver and sends it the DRV_QUERYCONFIGURE message. If a driver responds that it can be configured--that is, it supports a configuration dialog box to set such parameters as the COM port, the interrupt number, or input and output (I/O) port address--then the application enables the Setup button. If the user chooses the Setup button, the application sends a DRV_CONFIGURE message to the driver.
Add Drivers	Installs a new driver.
Default	Redisplays the list of files from the MMSETUP.INF file. Note that the Default button is active when the OEM drivers are displayed.

Installing a Driver

When the user selects a driver from the Installed Drivers list box, the Add Driver dialog box closes. The new driver becomes selected in the list box when the user chooses the OK button. The Drivers Control Panel application sends the DRV_INSTALL message to the driver if there is only one entry in the SYSTEM.INI file for it. (A driver receives the DRV_INSTALL message for its initial installation.) The Drivers Control Panel application can install up to four wave devices, four musical instrument digital interface (MIDI) devices, and ten media control interface (MCI) devices of the same type. If the selected driver is not an installable driver, the Driver Control Panel applications displays a "Cannot Install" message. If the user chooses the Cancel button, the dialog box closes with no changes made.

Using Drivers with the Drivers Control Panel Application

During installation, the Drivers Control Panel application opens the driver and obtains the description line, originally defined in the module-definition (.DEF) file, from the driver header. The application uses the description line to construct the settings for the [drivers] section. The description line in the .DEF file should have the following form:

```
DESCRIPTION type(s):text
```

Following are the parameters in the description line:

Parameter	**Meaning**
type(s)	Type of driver used for the entry in the SYSTEM.INI file. Multiple entries are separated by commas.
text	Text that describes the driver. This will be displayed in the Drivers Control Panel application.

For example, the header file for an oscilloscope driver (OSCI.DRV) can use the following description line:

```
DESCRIPTION 'FREQ,AMPL:Oscilloscope frequency and amplitude drivers.'
```

Based on this definition, if both drivers are installed (that is, if the Drivers Control Panel application displays a selection for both FREQ and AMPL), the Drivers Control Panel application creates the following settings in the SYSTEM.INI file:

.
.
.

```
[drivers]
FREQ = osci.drv
AMPL = osci.drv
.
.
.
```

If you want your driver added to a named section of the SYSTEM.INI file, you can add the section name to the type of driver. For example, the following description line specifies that a voltmeter driver be added to the [RCC] section:

```
DESCRIPTION 'VOLTMETER[RCC]:RCC voltmeter driver.'
```

Creating a Custom Configuration Application

The Drivers Control Panel application provides a convenient interface for installing drivers. You should use this interface for configuring features that are hardware- or driver-dependent. If your driver configures system features--those features that are hardware- and device-independent--you should create a custom Control Panel application.

E.2 Video Subsystem Drivers

Each video subsystem type can include one or more system drivers that perform overlay (ie. Draw Handlers, VidCap driver, hardware CODEC drivers, custom video-teleconferencing drivers, etc.). By default, the VAFC connector is in 8-bit palettized output mode. It is suggested that the VAFC driver does not assume **any** initial state of the system, and that it initialize the graphics subsystem to some default state. With this in mind, it should also not be assumed by the video subsystem driver what initial state the graphics subsystem is in, and if it needs to know, it should use the API to get the initial state.

It is the responsibility of these different video subsystem drivers that come with the video board to set the VAFC interface up to be able to use it. Clearly, each video subsystem driver can contain the code to enable and configure the interface or it can be centralized into one subsystem driver that the other drivers call. Regardless, once the connector (and both sides of the connector) is configured using the methods explained in Section 3.0 (pp. 5), the video subsystem can continue to perform whatever tasks it needs using the connector.

It should be noted that it is the responsibility of the video subsystem driver to enable multiple instances of its application to use the bus. It does the context switching. The context switching is not part of the VAFC driver.

With respect to the installation of the video subsystem drivers, each video subsystem driver has a different installation procedure which is beyond the scope of this document.

Appendix F: Color Space Formats

The following section briefly outlines the base color space formats described in this document and their respective data representation as they are transferred on the VAFC bus. For more information regarding the transfer of the pixel data relative to the clock, refer to the VAFC Hardware Specification Revision 1.0. The YCrCb formats should be encoded in standard CCIR 601 levels.

8-Bit RGB, Indexed

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Data	P7	P6	P5	P4	P3	P2	P1	P0

8-Bit RGB, 3:3:2

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Data	R2	R1	R0	G2	G1	G0	B1	B0

15-Bit RGB, 5:5:5

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Data	-	R4	R3	R2	R1	R0	G4	G3	G2	G1	G9	B4	B3	B2	B1	B0

15-Bit aRGB, 1:5:5:5

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Data	A0	R4	R3	R2	R1	R0	G4	G3	G2	G1	G9	B4	B3	B2	B1	B0

16-Bit RGB, 5:6:5

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Data	R4	R3	R2	R1	R0	R5	R4	R3	R2	R1	R0	B4	B3	B2	B1	B0

24-Bit RGB, 8:8:8

Bit	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Data	-	-	-	-	-	-	-	-	R7	R6	R5	R4	R3	R2	R1	R0

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Data	G7	G6	G5	G4	G3	G2	G1	G0	B7	B6	B5	B4	B3	B2	B1	B0

32-Bit aRGB, 8:8:8:8

Bit	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Data	A7	A6	A5	A4	A3	A2	A1	A0	R7	R6	R5	R4	R3	R2	R1	R0

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

Data	G7	G6	G5	G4	G3	G2	G1	G0	B7	B6	B5	B4	B3	B2	B1	B0

16-Bit YCrCb, 4:1:1

It takes four 4:1:1 16-bit pixels to complete the CbCr information for those four pixel.

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Data	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Cb1	Cb0	Cr1	Cr0	-	-	-	-

16-Bit YCrCb, 4:2:2

Bits 31-24 are the Y component for the second pixel, and bits 15-8 are the Y component for the first pixel.

Bit	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Data	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Cr7	Cr6	Cr5	Cr4	Cr3	Cr2	Cr1	Cr0

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Data	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Cb7	Cb6	Cb5	Cb4	Cb3	Cb2	Cb1	Cb0

24-Bit YCrCb, 4:4:4

Bit	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Data	X	X	X	X	X	X	X	X	Cr7	Cr6	Cr5	Cr4	Cr3	Cr2	Cr1	Cr0

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Data	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Cb7	Cb6	Cb5	Cb4	Cb3	Cb2	Cb1	Cb0